**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

ФАИС

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

по дисциплине **«Языки программирования высокого уровня»**

на тему: **«Использование графических классов .NET»**

Выполнил: студент гр. ИП-22

Коваленко А. И.

Принял: ст. преподаватель Романькова Т. Л.

Дата сдачи отчета: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

**Использование графических классов .NET**

**Цель**: *получить навыки разработки программ, использующих графические классы Pen, Brush,Graphics, GraphicsPath, Image и др.*

**Вариант 7**

Разработать приложение, позволяющее строить график исходной зависимости, заданной таблично и записанной в текстовый файл, в виде отдельных точек (маркеров) и график аппроксимирующей функции вида Ax2+Bx+C, коэффициенты которой получены методом наименьших квадратов.

**Код программы:**

using System;

using System.Windows.Forms;

namespace WindowsFormsApp1

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

private double xMin;

private double xMax;

private double step;

private double[] x;

private double[] y1;

private double[] y2;

Chart chart;

private List<MyPoint> list;

private double coefA;

private double coefB;

private double coefC;

private const double EXPANSION\_OF\_GRAPHIC = 20;

public Form1()

{

InitializeComponent();

(list, coefA, coefB, coefC) = SolutionOfEquation.TakeCoef();

xMin = list.Select(i => i.PointX).Min() - EXPANSION\_OF\_GRAPHIC;

xMax = list.Select(i => i.PointX).Max() + EXPANSION\_OF\_GRAPHIC;

step = (xMax - xMin) / EXPANSION\_OF\_GRAPHIC;

Form1\_Load();

}

private void CalcFunction()

{

double a, b, c;

(list, a, b, c) = SolutionOfEquation.TakeCoef();

// Количество точек графика

int count = (int)Math.Ceiling((xMax - xMin) / step)

+ 1;

// Создаѐм массивы нужных размеров

x = new double[count];

y1 = new double[count];

y2 = new double[count];

// Расчитываем точки для графиков функции

for (int i = 0; i < count; i++)

{

// Вычисляем значение X

x[i] = xMin + step \* i;

// Вычисляем значение функций в точке X

y1[i] = coefA \* x[i] \* x[i] + coefB \* x[i] + coefC; // - 0.25 \* x[i];

y2[i] = Math.Cos(x[i]);

}

}

private void CreateChart()

{

// Создаѐм новый элемент управления Chart

chart = new Chart();

// Помещаем его на форму

chart.Parent = this;

// Задаѐм размеры элемента

chart.SetBounds(10, 10, ClientSize.Width - 20,

ClientSize.Height - 20);

// Создаѐм новую область для построения графика

ChartArea area = new ChartArea();

// Даѐм ей имя (чтобы потом добавлять графики)

area.Name = "myGraph";

// Задаѐм левую и правую границы оси X

area.AxisX.Minimum = xMin;

area.AxisX.Maximum = xMax;

// Определяем шаг сетки

area.AxisX.MajorGrid.Interval = step;

// Добавляем область в диаграмму

chart.ChartAreas.Add(area);

// Создаѐм объект для первого графика

Series series1 = new Series();

// Ссылаемся на область для построения графика

series1.ChartArea = "myGraph";

// Задаѐм тип графика - сплайны

series1.ChartType = SeriesChartType.Spline;

// Указываем ширину линии графика

series1.BorderWidth = 3;

// Название графика для отображения в легенде

series1.LegendText = "A\*x^2+B\*x+C";

// Добавляем в список графиков диаграммы

chart.Series.Add(series1);

// Аналогичные действия для второго графика

Series series2 = new Series();

series2.ChartArea = "myGraph";

series2.ChartType = SeriesChartType.Spline;

series2.BorderWidth = 3;

series2.LegendText = "Points";

chart.Series.Add(series2);

// Создаѐм легенду, которая будет показывать названия

Legend legend = new Legend();

chart.Legends.Add(legend);

}

private void Form1\_Load()

{

// Создаѐм элемент управления

CreateChart();

// Расчитываем значения точек графиков функций

CalcFunction();

// Добавляем вычисленные значения в графики

double[] xPoint = list.Select(i => i.PointX).ToArray();

double[] yPoint = list.Select(i => i.PointY).ToArray();

chart.Series[0].Points.DataBindXY(x, y1);

chart.Series[1].MarkerColor = Color.Red;

chart.Series[1].Points.DataBindXY(xPoint, yPoint);

chart.Series[1].ChartType = SeriesChartType.Point;

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApp1

{

class FileWork

{

public static List<MyPoint> ToPointList(string fileName)

{

List<string> list = new List<string>();

using (StreamReader reader = new StreamReader(fileName))

{

for (int i = 0; i < File.ReadAllLines(fileName).Length; i++)

{

list.Add(reader.ReadLine());

}

}

List<MyPoint> e = new List<MyPoint>();

foreach (string i in list)

{

string[] tmp = i.ToString().Split(' ');

double x = Convert.ToDouble(tmp[0]);

double y = Convert.ToDouble(tmp[1]);

e.Add(new MyPoint(x, y));

}

return e;

}

}

}

namespace WindowsFormsApp1

{

class MyPoint

{

public double PointX { get; set; }

public double PointY { get; set; }

public MyPoint(double x, double y)

{

PointX = x;

PointY = y;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace WindowsFormsApp1

{

class SolutionOfEquation

{

public static (List<MyPoint> points, double a, double b, double c) TakeCoef()

{

List<MyPoint> list = FileWork.ToPointList(@"..\..\Points.txt");

double[][] m;

double[] b;

(m, b) = GetAmounts(list);

double[] x = SystemSolve(m, b);

return (list, x[0], x[1], x[2]);

}

public static (double[][], double[]) GetAmounts(List<MyPoint> list)

{

double amountX4 = list.Select(i => Math.Pow(i.PointX, 4)).Sum();

double amountX3 = list.Select(i => Math.Pow(i.PointX, 3)).Sum();

double amountX2 = list.Select(i => Math.Pow(i.PointX, 2)).Sum();

double amountX = list.Select(i => i.PointX).Sum();

double amountX2Y = list.Select(i => i.PointX \* i.PointX \* i.PointY).Sum();

double amountXY = list.Select(i => i.PointX \* i.PointY).Sum();

double amountY = list.Select(i => i.PointY).Sum();

double size = list.Count;

double[][] m = MatrixCreate(3, 3);

m[0][0] = amountX4; m[0][1] = amountX3; m[0][2] = amountX2;

m[1][0] = amountX3; m[1][1] = amountX2; m[1][2] = amountX;

m[2][0] = amountX2; m[2][1] = amountX; m[2][2] = size;

double[] res = { amountX2Y, amountXY, amountY };

return (m, res);

}

static double[] HelperSolve(double[][] luMatrix, double[] b)

{

// Решаем luMatrix \* x = b

int n = luMatrix.Length;

double[] x = new double[n];

b.CopyTo(x, 0);

for (int i = 1; i < n; ++i)

{

double sum = x[i];

for (int j = 0; j < i; ++j)

sum -= luMatrix[i][j] \* x[j];

x[i] = sum;

}

x[n - 1] /= luMatrix[n - 1][n - 1];

for (int i = n - 2; i >= 0; --i)

{

double sum = x[i];

for (int j = i + 1; j < n; ++j)

sum -= luMatrix[i][j] \* x[j];

x[i] = sum / luMatrix[i][i];

}

return x;

}

static double[] SystemSolve(double[][] A, double[] b)

{

// Решаем Ax = b

int n = A.Length;

int[] perm;

int toggle;

double[][] luMatrix = MatrixDecompose(

A, out perm, out toggle);

if (luMatrix == null)

return null; // или исключение

double[] bp = new double[b.Length];

for (int i = 0; i < n; ++i)

bp[i] = b[perm[i]];

double[] x = HelperSolve(luMatrix, bp);

return x;

}

//static double MatrixDeterminant(double[][] matrix)

//{

// int[] perm;

// int toggle;

// double[][] lum = MatrixDecompose(matrix, out perm, out toggle);

// if (lum == null)

// throw new Exception("Unable to compute MatrixDeterminant");

// double result = toggle;

// for (int i = 0; i < lum.Length; ++i)

// result \*= lum[i][i];

// return result;

//}

static double[][] MatrixDecompose(double[][] matrix, out int[] perm, out int toggle)

{

// Разложение LUP Дулитла. Предполагается,

// что матрица квадратная.

int n = matrix.Length; // для удобства

double[][] result = MatrixDuplicate(matrix);

perm = new int[n];

for (int i = 0; i < n; ++i) { perm[i] = i; }

toggle = 1;

for (int j = 0; j < n - 1; ++j) // каждый столбец

{

double colMax = Math.Abs(result[j][j]); // Наибольшее значение в столбце j

int pRow = j;

for (int i = j + 1; i < n; ++i)

{

if (result[i][j] > colMax)

{

colMax = result[i][j];

pRow = i;

}

}

if (pRow != j) // перестановка строк

{

double[] rowPtr = result[pRow];

result[pRow] = result[j];

result[j] = rowPtr;

int tmp = perm[pRow]; // Меняем информацию о перестановке

perm[pRow] = perm[j];

perm[j] = tmp;

toggle = -toggle; // переключатель перестановки строк

}

if (Math.Abs(result[j][j]) < 1.0E-20)

return null;

for (int i = j + 1; i < n; ++i)

{

result[i][j] /= result[j][j];

for (int k = j + 1; k < n; ++k)

result[i][k] -= result[i][j] \* result[j][k];

}

} // основной цикл по столбцу j

return result;

}

static double[][] MatrixDuplicate(double[][] matrix)

{

// Предполагается, что матрица не нулевая

double[][] result = MatrixCreate(matrix.Length, matrix[0].Length);

for (int i = 0; i < matrix.Length; ++i) // Копирование значений

for (int j = 0; j < matrix[i].Length; ++j)

result[i][j] = matrix[i][j];

return result;

}

static double[][] MatrixCreate(int rows, int cols)

{

// Создаем матрицу, полностью инициализированную

// значениями 0.0. Проверка входных параметров опущена.

double[][] result = new double[rows][];

for (int i = 0; i < rows; ++i)

result[i] = new double[cols]; // автоинициализация в 0.0

return result;

}

}

}

**Результаты работы программы:**

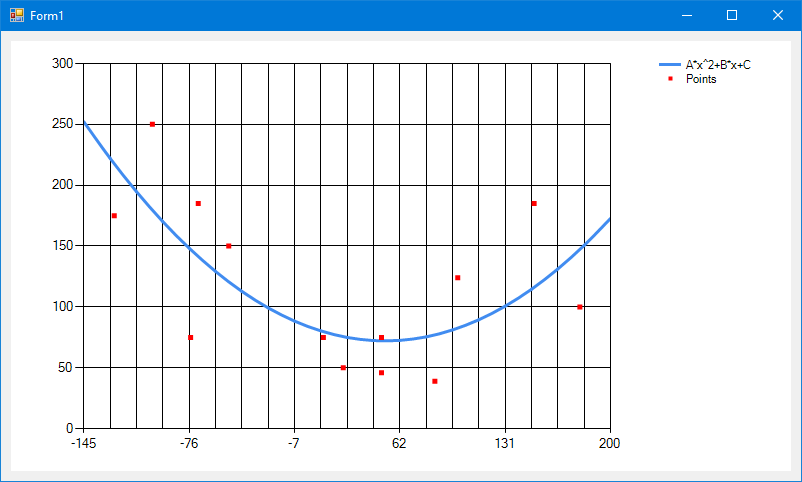


Рисунок 1 – результат работы программы

**Вывод:** *В ходе лабораторной работы были получены навыки разработки программ, использующих графические классы Pen, Brush,Graphics, GraphicsPath, Image и др..*